An Approach to AP Interoperability in a File Exchange Environment

by
Mitchell Gilbert

Introduction

A great deal of work and thought has taken place within PDES, Inc. concerning the ability of the STEP models to support a concept called "AP Interoperability". Many discussions have been held to consider the meaning of this term and the requirements for both the standard and implementations of the standard in order to achieve this goal. For the purposes of this paper a narrow focus of this concept is going to be discussed. That focus is the need to be able to develop application software for physical file exchange using more than one application protocol together in concert.

In previous discussions a basic assumption was agreed to among all participants. That assumption is that it is unreasonable to expect that an implementation of one AP will be able to process a physical file produced by an implementation of another AP without the knowledge of that other APs schema. This simple understanding provides a basis for an implementation architecture for STEP. This piece of the STEP data sharing puzzle is one which has only now being explored. Once a viable implementation architecture becomes well defined and understood along with the STEP model architecture, data sharing will become an achievable goal.

2 STEP Method and Architecture

STEP is a standard which is based on conceptual modeling as one of its foremost principles. As such, STEP has used an information modeling language in order to specify the meaning of the product data which is being represented and exchanged. A basic principle in the use of this information modeling language has been the development of a data architecture and methodology that carefully balances the common thread of the standard, product data, with different views of the common thread based on the recognition of differing requirements depending on differing uses. This common thread is represented in the STEP architecture by two different types of constructs. The first type of construct is the integrated resource construct. These constructs are very broad and generic in that they are available to satisfy the requirements of many different applications. The integrated resources form building blocks with which the application's requirements can be fulfilled. The process of fulfilling these requirements is called interpretation. During this process, very generic concepts are used, specialized or constrained in order to satisfy the requirements of a particular application.

The second type of construct is the Application Interpreted Construct or AIC. The AICs are a recognition in the STEP architecture that some interpretations as defined above may belong to more than one AP. that is, the same set of requirements for specialization and constraint exist within more than one AP. The AIC development process will define a STEP conceptual construct or set of constructs that will represent and bound that common interpretation. The definition of the AIC, however, must involve the application of some AP's requirements against the integrated resources.

The recognition of differing requirements is manifested in STEP as the Application Protocol and, therefore, the application interpreted model. The application interpreted model defines the constructs to be implemented, both the common constructs to product data and the

application specific constructs. These requirements are painstakingly developed by the application protocol development team. They develop an application activity model of a process that drives the discovery of data. That process is then detailed into an information model called the application reference model and validated using experts in the application area. The application reference model is then driven into the application interpreted model by mapping the requirements of the application to the generic product data defined in the integrated resources and application interpreted constructs using a strict set of rules about what may and may not be done. These application interpretation rules for a well defined boundary between the common thread of STEP and the application specific thread of STEP.

3 The Problem

Since all of the application interpreted models are built using these common building blocks, the data which they define may be shared. This data may be application specific in the case of shared AIC structure among different APs, or it may be generic in nature, and achieved by the use of common integrated resource constructs. Because all APs application interpreted models share these common bonds, there are some specific questions that arise when we think about the interoperability of two or more APs.

1. What is the common data?

2. What are the common concepts?

Once we answer these questions, we can have meaningful interoperability among different applications which use different APs. The problem that we face whenever we discuss interoperability is that it becomes a rote process, instead of a meaningful one. The questions above are not answered, and oftentimes may not even be asked. The question becomes one of structure. The questions become more like:

1. What is the common structure?

2. What is interfering with the sharing of that structure?

3. How can I ram a physical file written in accordance with one AP into my implementation of another AP and derive meaningful information?

If we ask these questions, then we are destined to fail in the quest of data sharing at a physical file level using STEP APs. Only by understanding the meaning of the information which is being exchanged and by planning to share common data, and therefore common concepts, can we achieve the type of data sharing that we have grown to expect from STEP implementations. The problem with asking the questions about common structure is that these questions circumvent the primary basis for STEP, the conceptual model. Of course the structure of the model is important, and we need to know the structure of the model, especially the common structure among APs in order to achieve data sharing, but that is not enough. Constraints written in both application interpreted models guide the allowable populations of the attributes of each AP. These constraints establish the relationships among the different structures and the standardize the interpretations of some attributes in a more concise way than the integrated resources can provide. We need to understand the relationships among the constructs that comprise the structure of the APs, and the meaning of the different structures given by entity structure, normative text and constraints in order to achieve useful data sharing.

4 Interoperability scenarios

To illustrate some of these data sharing concepts, an EXPRESS-G model of some of the common concepts among APs is given as figure 1. This model is not one of any particular AP in STEP. It does, however, provide a similar conceptual representation to the information found in the application interpreted models

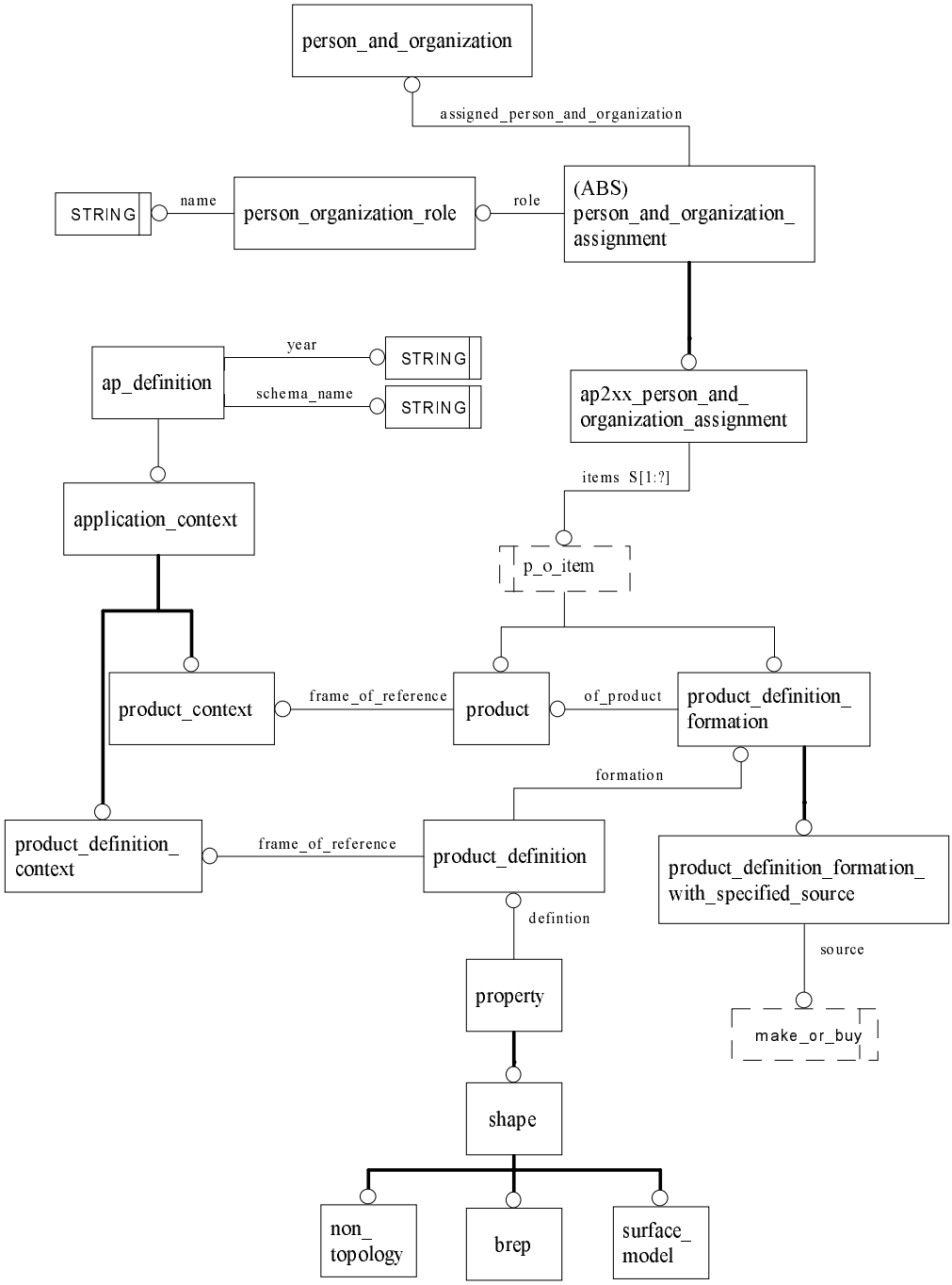

Figure 1 - EXPRESS-G Example Model

of both AP202 and AP203. The model has been simplified for the purpose of illustration. Since the model has been simplified, to remove confusion, the two APs will simply be discussed as AP1 and AP2. The model includes the definition of a product, the versioning of that product, the definition of the product, one property of the product - shape, three different representation types for that shape and the association of person and organization information with the product and the version. What is not shown in the EXPRESS-G is the constraints in the model and a couple of distinguishing factors between the two AP schemas.

The two AP schemas will each contain a rule on the name attribute of the person_organization_role entity. For discussion's sake, lets say that in AP1, the rule says that the name attribute for person_organization_role shall have a value of either "a", "b" or "c". The rule on the same attribute in the AP2 schema specifies that the attribute shall have a value of either "c", "d", or "e". These rules are written in accordance with the interpretation method for the enumeration of STRING typed attributes. That method specifies that a global rule is written in each schema for the person_organization_role entity that constrains each instance of that entity created within the scope of each schema to have a name attribute with the previously defined strings. Furthermore, there are some differences in the scope of each AP model. The schema shown here is equivalent to the union of the two AP schemas. In AP1, the subtype product_definition_formation_with_specified_source of product_definition_formation is out of scope. The supertype, product_definition_formation, is within the scope of AP1. AP2, on the other hand, has a requirement for a product_definition_formation with the source information specified and therefore has a constraint that within its scope all product_definition_formation entities must specify their source and the subtype entity must always be used. The other difference in the two APs is that the
AP2XX_person_and_organization_assignment entity is different in each schema, and the select type p_o_item may select different entities which are not given in figure 1.

There are two important aspects of the model which need to be discussed before examining the issues that have been raised concerning data sharing. The two important aspects are the product_context and the product_definition_context. Since the STEP standard is about **product data**, every data exchange that conforms to any STEP AP will contain at least an instance of the product entity. In order to specify some information that defines different aspects of the product such as properties or presentation the concept of a product's definition must be used and therefore those data exchanges will contain instances of the product_definition entity. The important concept to notice is that the model contains navigation paths so that any instance of any entity may always be traced back to its context within STEP. The context, then, will contain data that specifies at least one ap_definition. This is true because each AP contains a rule that specifies that there must be exactly one ap_definition that calls out the AP schema name referencing the application_context entity. So, we see that every entity instance in a physical file can be traced back to its AP meta-data.

With this in mind, and keeping in mind our original assumption that the processors must understand the syntax and semantics (AP schema) of another AP in order to understand the common data, let's now examine some data sharing problems and some possible solutions which can be implemented. Four scenarios will be discussed:

1. How does an application share data when the structure and meaning is exactly the same.

2. How does an application conforming to AP1 share data with another application conforming to AP2 if one AP has a more generic scope than the other.

3. How does an application conforming to AP1 share data with another application conforming to AP2 if there are different, possibly conflicting, constraints on the population of the attributes of the same entity.
4. How does an application conforming to AP1 share data with another application conforming to AP2 if there are different entities for the assignment of management resources (like person_and_organization).

The first scenario consists of the sharing of data when the schemas are exactly the same. The area in the example is that of product_definition, property, shape and the different representations of shape. Since the physical file instances for these entities are exactly the same, a processor for AP1 and for AP2 will have little trouble understanding these constructs. The application code should be exactly the same to process these types of things. In addition, since the representation type entities are derived from a common basis, an AIC, there is a shared interpretation here. The constraints on the entities within the AICs whose scope is defined by these entities are the same and therefore will be enforced in the same manner across all APs which use them.

The second scenario entails the sharing of the common data when one AP has a more generic scope than another AP. This scenario is illustrated by the use of product_definition_formation and product_definition_formation_with specified_source. AP1 uses only the more generic supertype and AP2 uses the more specific subtype in order to specify the source of the product_definition_formation. Furthermore, as previously discussed, AP2 constrains the product_definition_formation to say that the only type meaningful to an application developed for its context is one in which the source is specified. The problem as stated is that the physical file "flattens" out the subtype/supertype structure so that the file processors will not see the structure. So a physical file which is written in accordance with AP1 will specify:

#1=PRODUCT_DEFINITION_FORMATION(of_product);

A physical file written in accordance with AP 2 will specify:

#1=PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE(of_product, source);

These entities are clearly different and, without the knowledge of the AP2 schema, a processor for AP1 would have no way of knowing that the underlying concept of product_definition_formation is the same in both cases. So, the only way to make this work is that an AP1 processor has to know the AP2 schema and the relationship between the two entities. Is this a one time solution? The answer is no. If an implementor wants to build up a system that will enable the systematic implementation of many different application protocols he can build a data dictionary capable of recognizing these relationships and reuse that data dictionary as more APs of interest to his customers are developed. Since the STEP architecture and methodology provides a common basis for all product data, all APs are GUARANTEED to build upon that common basis.

There is another issue in this scenario, and that issue is loss of data or lack of data. If an AP1 processor is reading a file written by an AP2 processor, there will be a loss of the source data. This, however, is not a problem since the application has no need for this information

and is only concerned with its particular function. The integration problem here is to develop compartments in which to store instances of the different, but related data. If an AP2 implementation receives a file from an AP1 implementation, then that application's user must, at some point, recognize that the source information is not specified in the AP1 file. The AP2 implementor may want to notify the user in some manner that there is incomplete information in the file due to the non-conformance to the AP1 rule that mandates the use of the subtype with source information.

The third scenario consists of the specification of differing constraints on the same entities in the different AP schemas. The basic concepts are shared, but the valid populations of those concepts are different, possibly conflicting, and possibly overlapping. The example in figure 1 is the name attribute of the person_organization_role entity. As previously discussed, AP1 contains a rule which says the only roles of interest within its scope are "a", "b", and "c". The roles of interest to AP2 are "c", "d", and "e". We can see quite clearly that the only role which the two APs have in common is "c". Therefore a physical file containing an instance of the person_organization_role entity with that role is of interest to both APs and the processors should be able to read and understand that role name. Otherwise the processors may do whatever they like with any non-conforming roles. The structure is there, however, to understand the concept and populate the data structure.

The real question here, is not whether the data is sharable, but what a processor will do when it recognizes invalid data. This is a design decision. If the application is integrated, one that manipulates data within the scope of BOTH AP1 and AP2, then it can look for any roles defined by either of the two APs. If the application is only built to process data according to a single AP then it may choose to ignore and flag data it finds invalid, that is in violation of the rule for the particular schema, or it may choose to process and store it. In any event, as demonstrated previously, the AP that created and "owns" the particular instance of person_organization_role is always able to be determined due to the existence dependence of all data in STEP on the AP meta-data. In order for an application to be more efficient, it may choose to keep track of that meta-data within the data dictionary for his particular application. A counter argument to this is that the constraint may be hidden in the AP2xx_person_and_organization_assignment entity. From a mechanical standpoint this is certainly possible. The semantics specified, however, are not what is intended. Instead of giving an enumeration for **all** instances of person_organization_role, the constraint will be specifying that all person_organization_role entities assigned to a piece of product data using AP2xx_person_organization_assignment shall have the desired roles. This is now no longer a constraint on the role, but one on the use of the role.

The fourth scenario is one in which each AP has a different subtype in its schema for the association of product data management concepts (like person_and_organization) with the product data. In this scenario, implementations of one AP1, knowing the schema of the other AP can process the associations due to the uniform method used across the breadth of APs. The information, however, is very application specific. Only in the context of the application and the rest of the application interpreted model does the information make sense. When product management data is, in fact, used generically, the management data itself is referenced through an attribute in the generic product data entity. Of course an AIC may be created when the association is completely shared among two or more application interpreted models and constraints which are dependent on that association are written into the association entity.

5 Conclusion

In this paper we have briefly explored the STEP model architecture, the STEP interpretation methodology and a possible STEP implementation strategy. We have also taken a look at some of the concepts within both AP202 and AP203 using some less complicated constructs. Four scenarios for interoperability have been presented with implementation solutions given for each. A basic observation which can be made is that STEP has a problem. The problem is that of education and experience. PDES, Inc. has made great strides in the development of STEP to provide an architecture that has a common basis for all applications that use product data. The same great strides can be made in data sharing utilizing these concepts given the proper planning and implementation strategy of APs in combination. The problem of education is great.

Application developers must be aware of the details of the STEP model architecture and semantics and the STEP development methodology in order to maximize the utility of their customer's ability to share data. PDES, Inc. has taken great care to ensure that the STEP standard takes an intelligent approach to product data. We must take the same care to ensure that the implementations that we, as users, demand from our application developers are intelligent and the applications that we, as vendors develop have that intelligence. It won't happen automatically, though. Education is our responsibility. Without the knowledge of the implementation strategy developed at the PDES, Inc. interoperability workshop and the data model knowledge an approach to system integration will be extremely difficult. As the knowledge increases, however, solutions to the problems will be workable and forthcoming.